EXHIBIT 5

Maker Docs

Median - Detailed Documentation

The Maker Protocol's trusted reference price

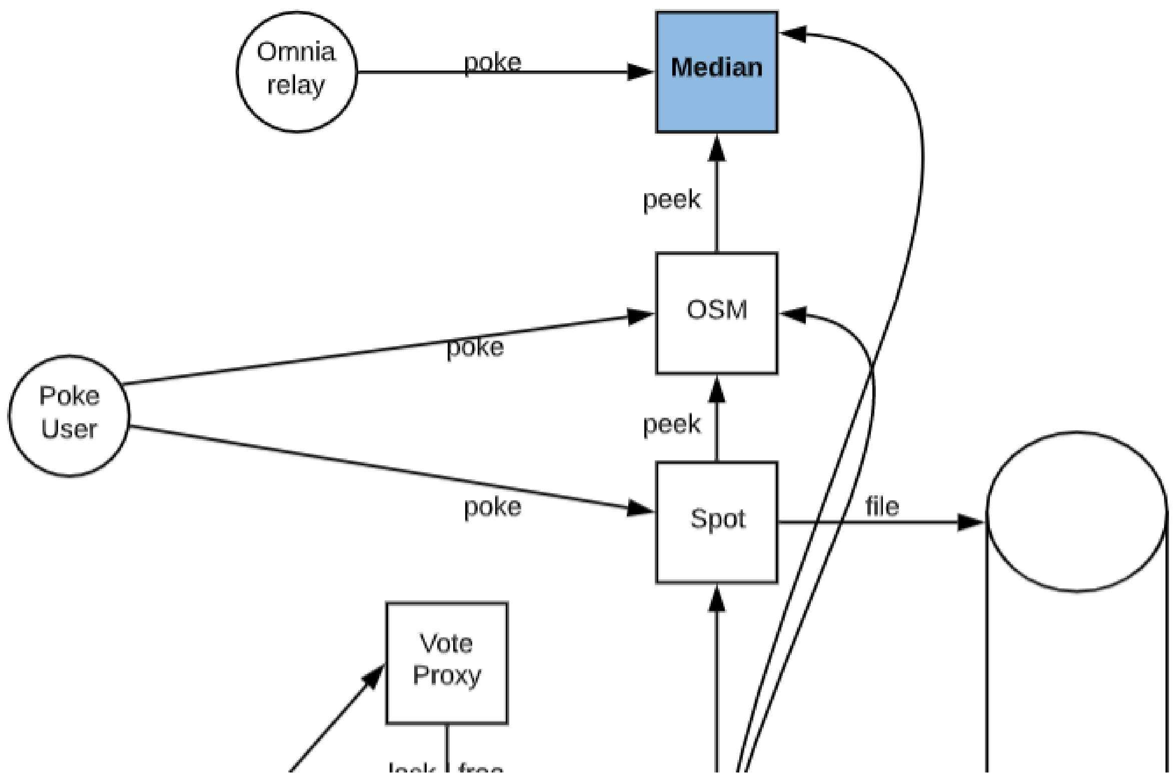
- **Contract Name:** median.sol
- **Type/Category:** Oracles Module
- **Associated MCD System Diagram**
- **Contract Source**

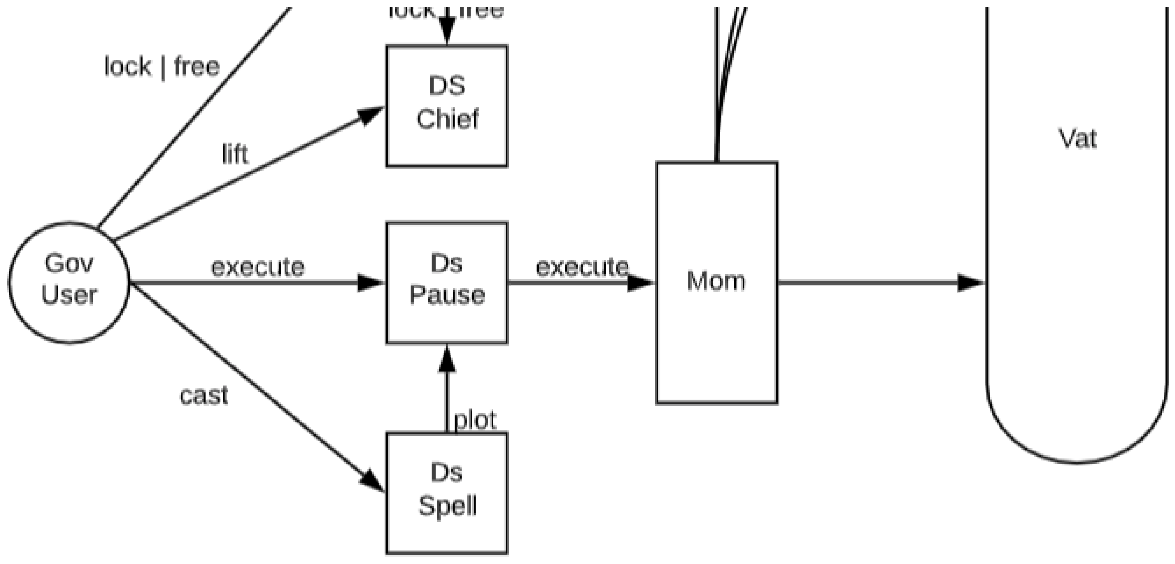
---

1. Introduction (Summary)

The median provides Maker's trusted reference price. In short, it works by maintaining a whitelist of price feed contracts which are authorized to post price updates. Every time a new list of prices is received, the median of these is computed and used to update the stored value. The median has permissioning logic which is what enables the addition and removal of whitelisted price feed addresses that are controlled via governance. The permissioning logic allows governance to set other parameters that control the Median's behavior—for example, the `bar` parameter is the minimum number of prices necessary to accept a new median value.

A High-level overview diagram of the components that involve and interact with the median:





Note: All arrows without labels are governance calls.

---

2. Contract Details

Median (Glossary)

Key Functionalities (as defined in the smart contract)

- `read` - Gets a non-zero price or fails.
- `peek` - Gets the price and validity.
- `poke` - Updates price from whitelisted providers.
- `lift` - Adds an address to the writers whitelist.
- `drop` - Removes an address from the writers whitelist.
- `setBar` - Sets the `bar`.
- `kiss` - Adds an address to the reader's whitelist.
- `diss` - Removes an address from the readers whitelist.

Note: `read` returns the `value` or fails if it's invalid & `peek` gives back the `value` and if the `value` is valid or not.

Other

- `wards(usr: address)` - Auth mechanisms.
- `orcl(usr: address)` - `val` writers whitelist / signers of the prices (whitelisted via governance / the authorized parties).
- `bud(usr: address)` - `val` readers whitelist.
-

`val` - the price (private) must be read with `read()` or `peek()`

- `val` - the price (private) must be read with `read()` or `peek()`
- `age` - the Block timestamp of last price `val` update.
- `wat` - the price oracles type (ex: ETHUSD) / tells us what the type of asset is.
- `bar` - the Minimum writers quorum for `poke` / min number of valid messages you need to have to update the price.

---

3. Key Mechanisms & Concepts

As mentioned above, the `median` is the smart contract that provides Maker's trusted reference price. Authorization **(auth)** is a key component included in the mechanism of this contract and its interactions. For example, the price (`val`) is intentionally kept not public because the intention is to only read it from the two functions `read` and `peek`, which are whitelisted. This means that you need to be authorized, which is completed through the `bud`. The `bud` is modified to get whitelisted authorities to read it on-chain (permissioned), whereas, everything of off-chain is public.

The `poke` method is not under any kind of `auth`. This means that anybody can call it. This was designed for the purpose of getting Keepers to call this function and interact with Auctions. The only way to modify its state is if you call it and send it valid data. For example, let's say this oracle needs 15 different sources. This means that we would need it to send 15 different signatures. It will then proceed to go through each of them and validate that whoever sent the the data has been `auth`'d to do so. In the case of it being an authorized oracle, it will check if it signed the message with a timestamp that is greater than the last one. This is done for the purpose of ensuring that it is not a stale message. The next step is to check for order values, this requires that you send everything in an array that is formatted in ascending order. If not sent in the correct order (ascending), the median is not calculated correctly. This is because if you assume the prices are ordered, it would just grab the middle value which may not be sufficient or work. In order to check for uniqueness, we have implemented the use of a `bloom` filter. In short, a bloom filter is a data structure designed to tell us, rapidly and memory-efficiently, whether an element is present in a set. This use of the bloom filter helps with optimization. In order to whitelist signers, the first two characters of their addresses (the first `byte`) have to be unique. For example, let's say that you have 15 different price signers, none of the first two characters of their addresses can be the same. This helps to filter that all 15 signers are different.

Next, there are `lift` functions. These functions tell us who can sign messages. Multiple messages can be sent or it can just be one but they are put into the authorized oracle). However, there is currently nothing preventing someone from `lift`'ing two prices signers that start with the same address. This is something for example, that governance needs to be aware of (see an example of what a governance proposal would look like in this case in the **Gotchas** section).

Due to the mechanism design of how the oracles work, the **quorum** has to be an odd number. If it is an even number, it will not work. This was designed as an optimization (`val = uint128(val_[val_.length >> 1]);`); this code snippet outlines how it works, which is by taking the array of values (all the prices that each of the prices signers reported, ordered from 200-215) and then grabbing the one in the middle. This is done by taking the length of the array (15) and shifting it to the right by 1 (which is the same as dividing by 2). This ends up being 7.5 and then the EVM floors it to 7. If we were to accept even numbers this would be less efficient. This presents the issue that you should have a defined balance between how many you require and how many signers you actually have. For example, let's say the oracle needs 15 signatures, you need at least 17-18 signers because if you require 15 and you only have 15 and one of them goes down, you have no way of modifying the price, so you should always have a bit more. However, you should not have too many, as it could compromise the operation.

---

4. Gotchas

Emergency Oracles

- They can shutdown the price feed but cannot bring it back up. Bringing the price feed back up requires governance to step in.

Price Freeze

- If you void the oracles Ethereum module, the idea is that you cannot interact with any Vault that depends on that ilk.
  - **Example:** ETHUSD shutdown (can still add collateral and pay back debt - increases safety) but you cannot do anything that increases risk (decreases safety - remove collateral, generate dai, etc.) because the system would not know if you would be undercollateralized.

Oracles Require a lot of Upkeep

- They need to keep all relayers functioning.
- The community would need to self-police (by looking at each price signer, etc.) if any of them needs to be replaced. They would need to make sure they are constantly being called every hour (for every hour, a transaction gets sent to the OSM, which means that a few transactions have already been sent to the median to update it as well. In addition, there would need to be a transaction sent to the `spotter`, as DSS operates in a pool-type method (doesn't update the system/write to it, you tell it to read it from the OSM).

There is nothing preventing from `lift`'ing two prices signers that start with the same address

- The only thing that this prevents is that you cannot have more than 256 oracles but we don't expect to ever have that many, so it is a hard limit. However, Governance needs to be sure that whoever they are voting in anyone that they have already voting in before with the same two first characters.
- An example of what a governance proposal would look like in this case:
  - We are adding a new oracle and are proposing (the Foundation) a list of signers (that have been used in the past) and we already have an oracle but want to add someone new (e.g. Dharma or dydx). We would say that they want to be price signers, so these are their addresses and we want to lift those two addresses. They would vote for that, and we would need to keep a list of the already existing addresses and they would need to create an address that doesn't conflict with the existing ones.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- By design, there is currently no way right now to turn off the oracle (failure or returns false) if all the oracles come together and sign a price of zero. This would result in the price being invalid and would return false on `peek`, telling us to not trust the value.
    - We are currently researching (Oracles ETH module) that would invalidate the price but there is no way to do this in the median today. This is due to the separation of concerns that DSS does not read directly from median, it reads from the OSM, but this may end up changing.